JIT Hooking

Hooking Just-in-Time
compiler of .NET Applications

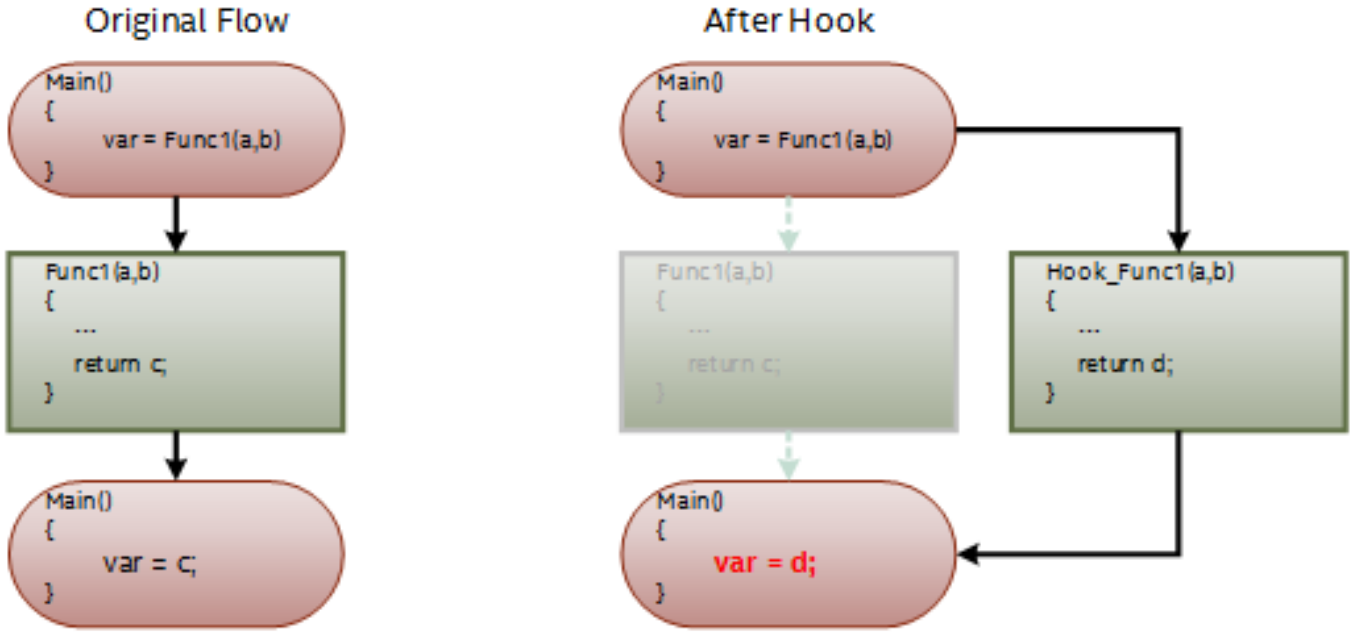
Reversing

O que é *hooking*?

- Técnica utilizada para modificar o comportamento de programas ou sistema operativo
  - Modificação em *runtime*
  - Conceito comum em *game hacking*, em particular no desenvolvimento de *cheats/hacks*

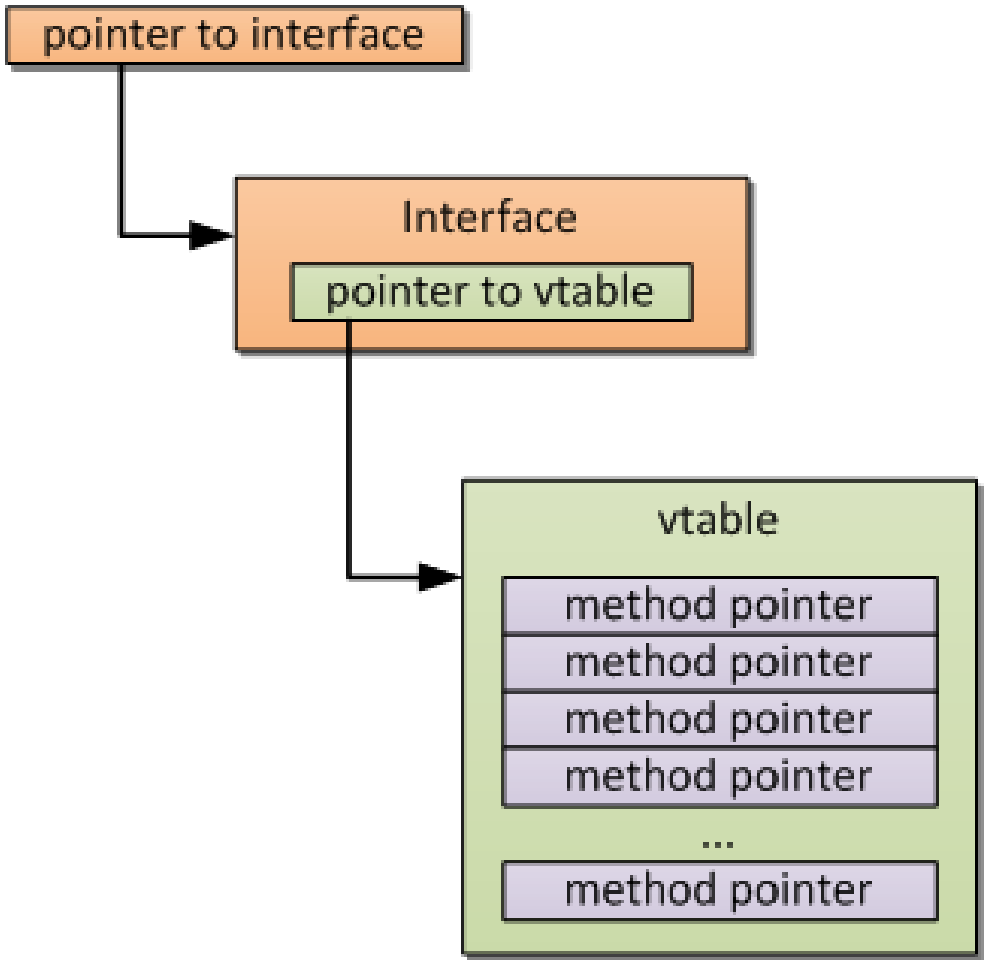
- Interceta chamadas a funções, mensagens ou eventos entre componentes

- *E.g.* Frida



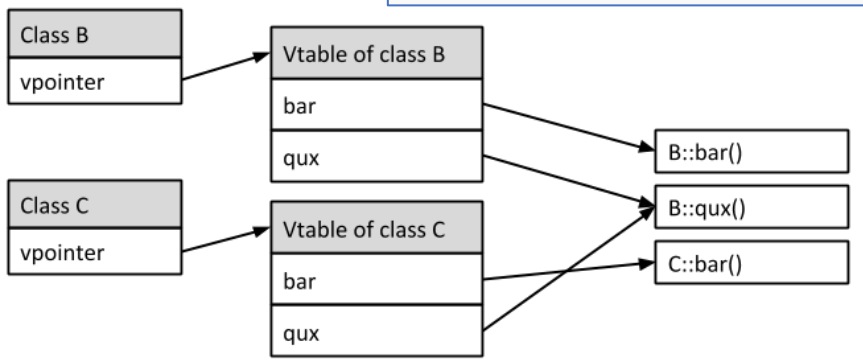
Técnicas de hooking

- VMT hooking
  - Substituir pointers na Virtual Method Table
- Inline/splice hooking
  - Overwrite primeiros X bytes com instr. JMP
  - Trampoline



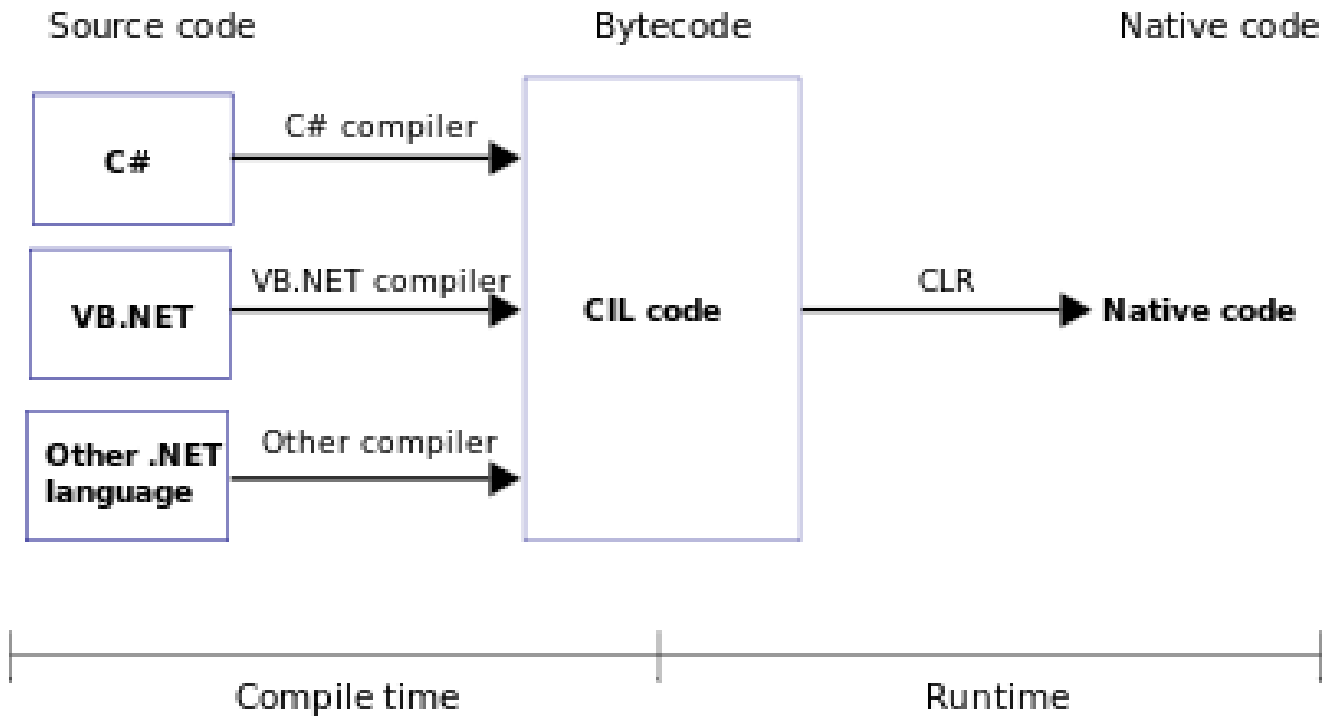

```cpp
#include <iostream>

class B
{
public:
  virtual void bar();
  virtual void qux();
};

void B::bar()
{
  std::cout << "This is B's implementation of bar" << std::endl;
}

void B::qux()
{
  std::cout << "This is B's implementation of qux" << std::endl;
}
```

```cpp
class C : public B
{
public:
  void bar() override;
};

void C::bar()
{
  std::cout << "This is C's implementation of bar" << std::endl;
}
```

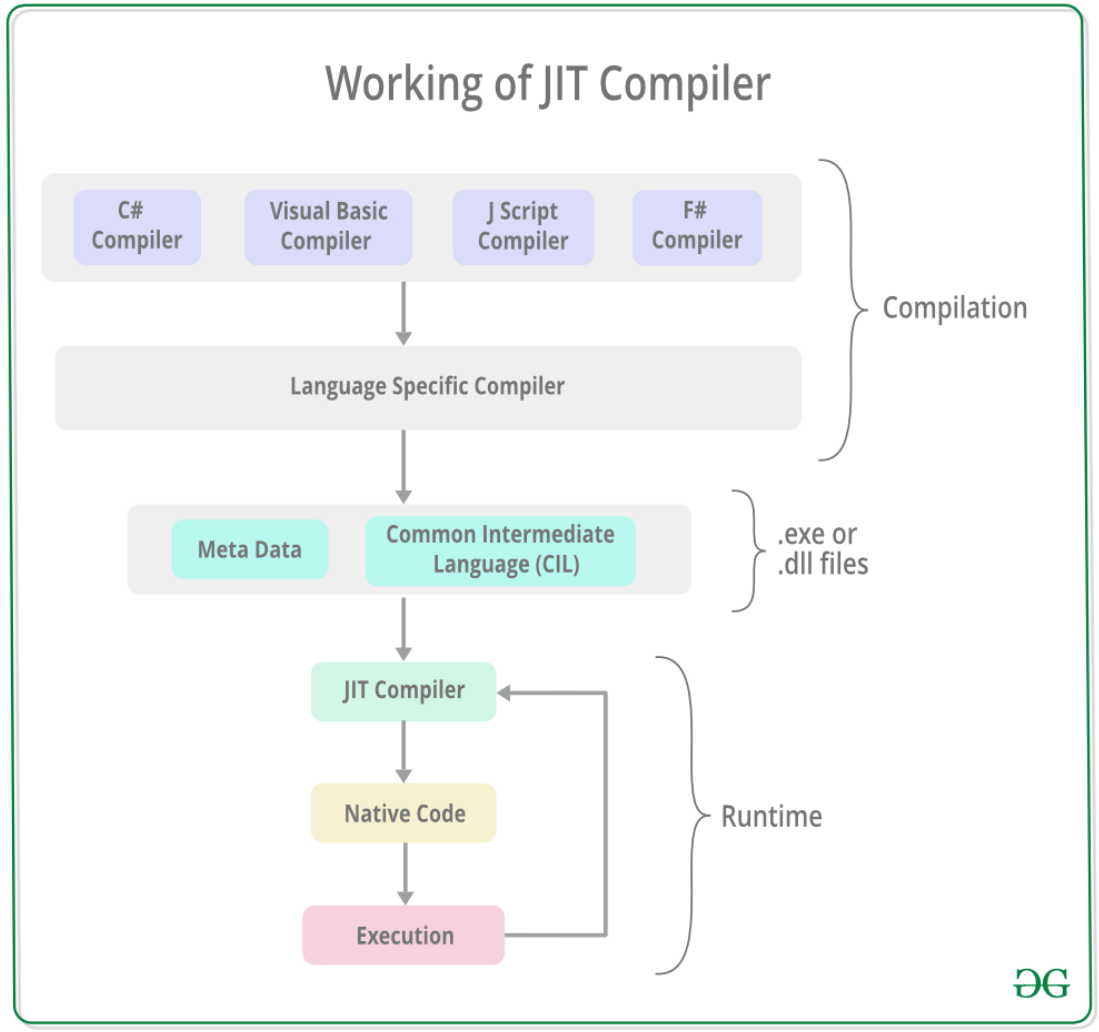


O que é *Just-in-Time*?

- Método de compilação
- Realiza a compilação durante execução do programa
  - *byte code (IR) → native code*
- *Dynamic Translation*

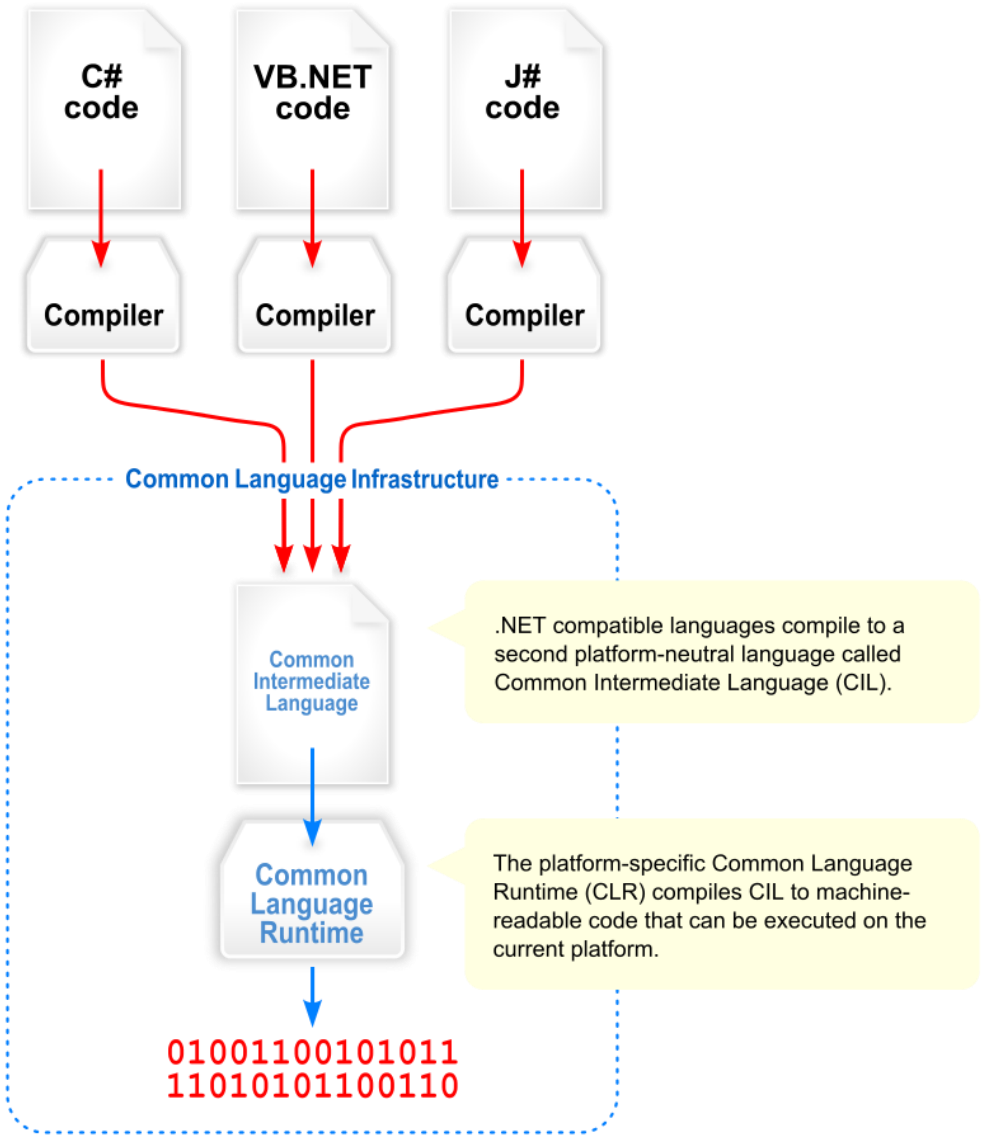


O que é *Just-in-Time*?



- *The JIT compiler reads the bytecodes in many sections (or in full, rarely) and compiles them dynamically into machine code so the program can run faster.*

O que é *Just-in-Time*?



- *The JIT compiler reads the bytecodes in many sections (or in full, rarely) and compiles them dynamically into machine code so the program can run faster.*

Common Language Infrastructure (CLI)

- Especificação/Standard originalmente desenvolvido pela Microsoft
  - ISO/IEC 23271 e ECMA 335

- Descreve código executável e um ambiente *runtime* que permite a utilização de várias linguagens de programação em diferentes plataformas
  - *Platform agnostic*

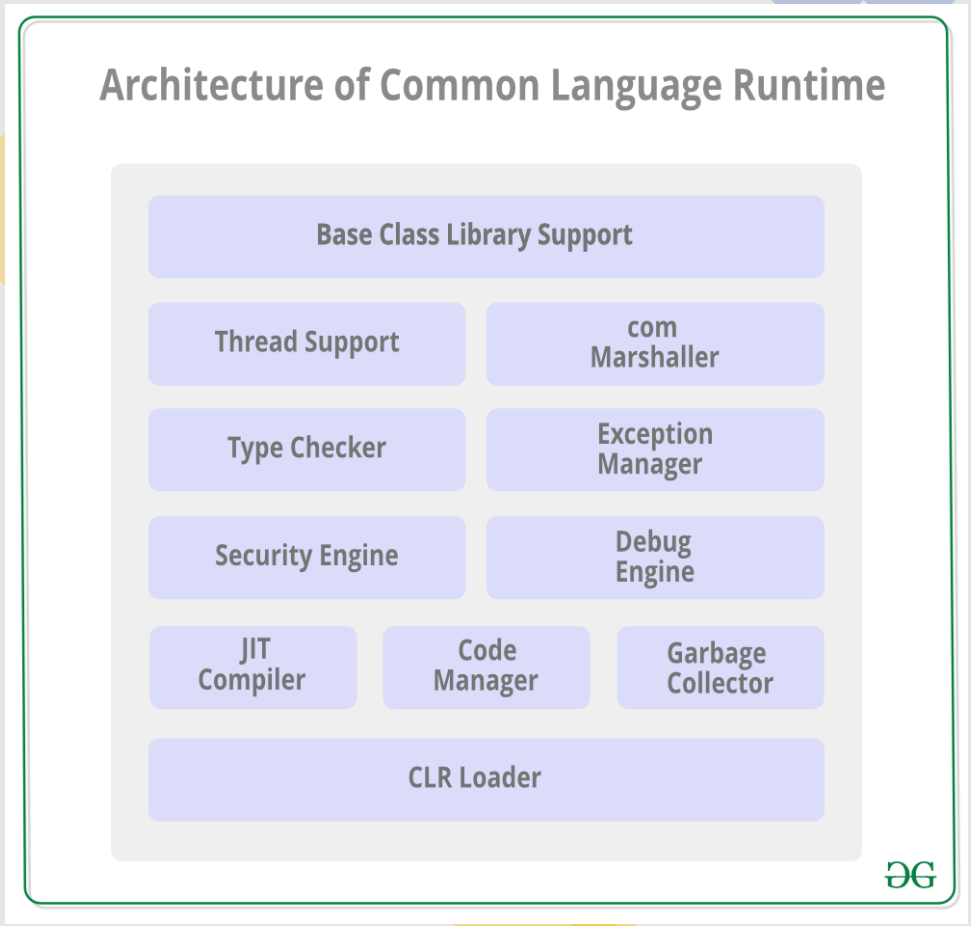
- Portabilidade, multi-plataforma

Common Language Infrastructure (CLI)

Implementações:
• .NET Framework (Commercial, superseded by .NET in 2020, ".NET 5") • .NET (Open-source multi-platform, sucessor of .NET Framework)   • .NET Core (https://github.com/dotnet/runtime/) • Mono (Alternative implementation)

Linguagens:
• C# • Visual Basic (VB.NET) • C++/CLI

Common Language Runtime (CLR)



Hooking .NET JIT Compiler

Aplicação .NET

- Módulos **Clrjit.dll** ou **Mscorjit.dll**

Clrjit.dll / Mscorjit.dll

- Neste módulo, existe uma função **getJit()** que retorna um pointer para uma interface **ICorJitCompiler**
- Responsável pela tradução
  - *byte code → native code*

TLDR;

- ICorJitCompiler* getJit();
- ICorJitCompiler::compileMethod(...);

Source “corjit.h”

```
// https://raw.githubusercontent.com/dotnet/runtime/main/src/coreclr/inc/corjit.h
class ICorJitCompiler;
class ICorJitInfo;

extern "C" ICorJitCompiler* getJit();

class ICorJitCompiler
{
public:
    // compileMethod is the main routine to ask the JIT Compiler to create native code for a method (...)
    virtual CorJitResult compileMethod (
            ICorJitInfo                 *comp,              /* IN */
            struct CORINFO_METHOD_INFO  *info,              /* IN */
            unsigned /* code:CorJitFlag */   flags,         /* IN */
            uint8_t                     **nativeEntry,      /* OUT */
            uint32_t                    *nativeSizeOfCode   /* OUT */
            ) = 0;
```

Objetivo

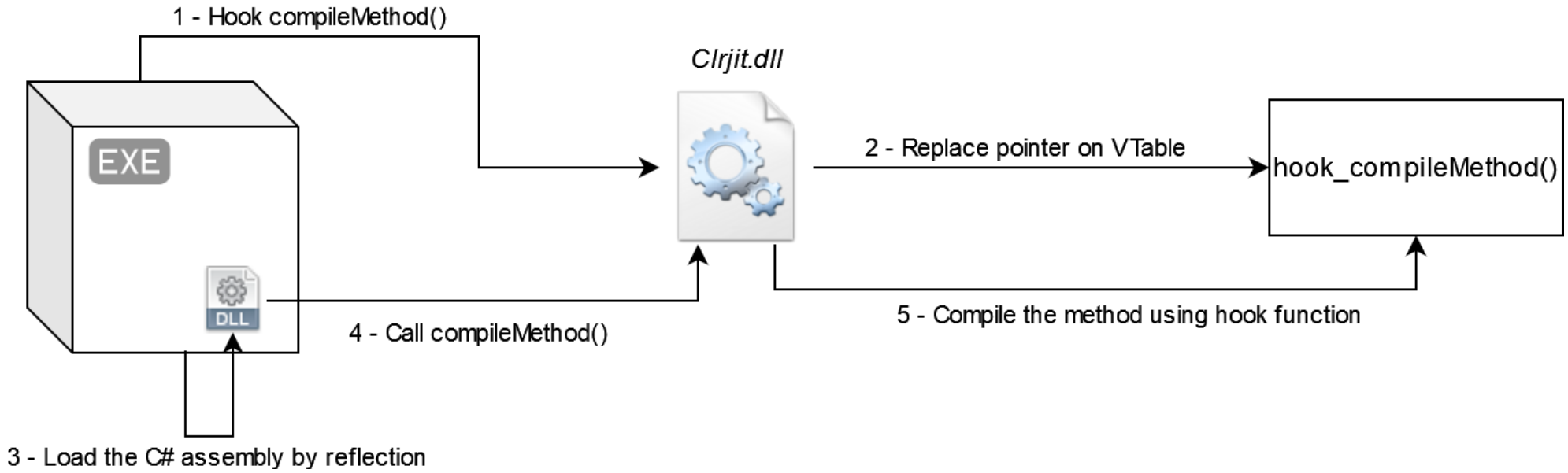
- Aplicação .NET (*e.g.* C#) cujos métodos vão ser inspecionados & modificados durante execução
- Programa C++/CLI que carrega a aplicação .NET e realiza o *hook* ao JIT:
  - Carrega a aplicação .NET (*Class Library* aka DLL)
  - Realiza o *hook* alterando o *pointer* de compileMethod() na VTable para outra função
  - Através da nova função, interceta e modifica código intermédio (IL) antes de ser *JITted*

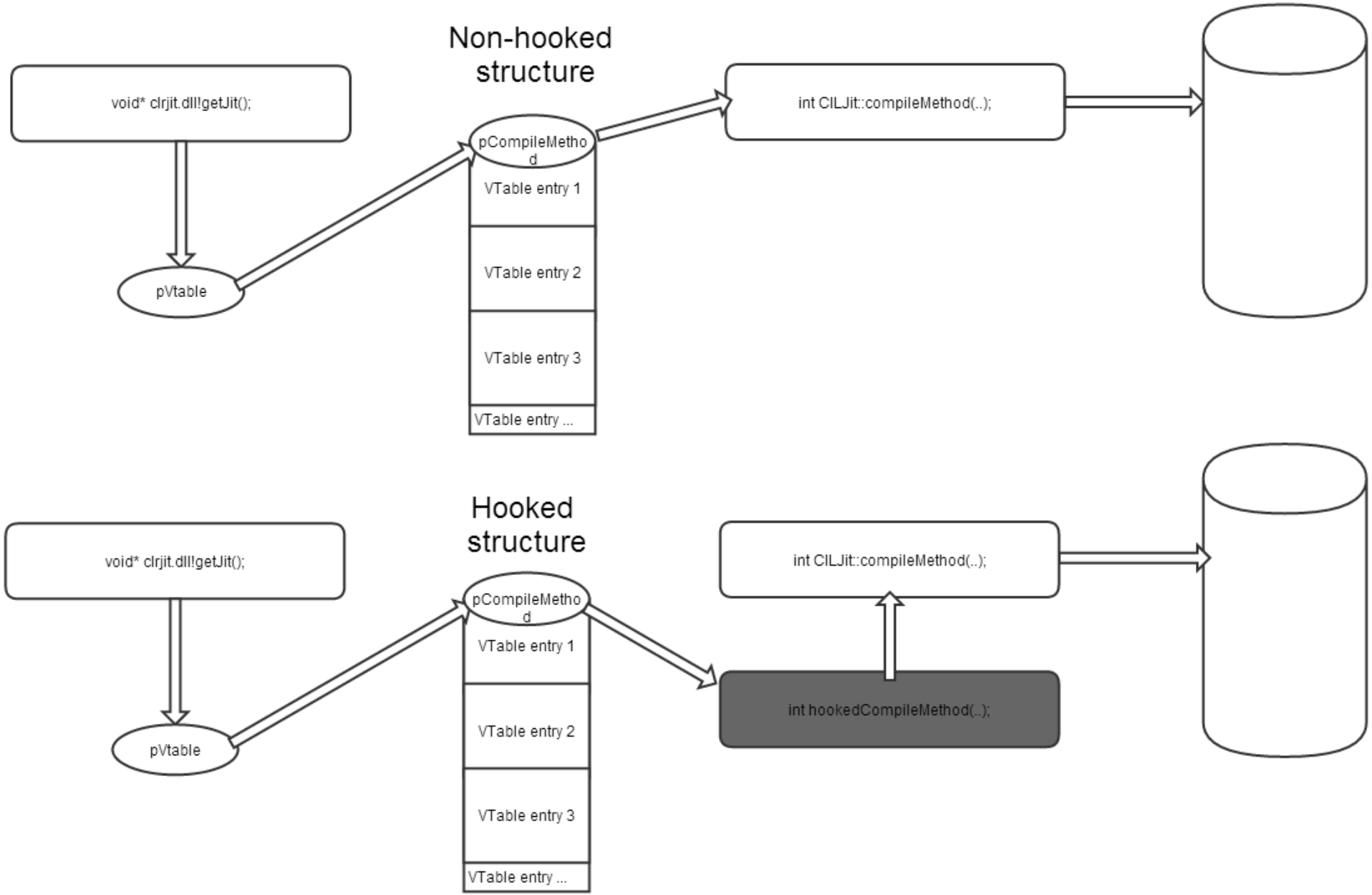
Ideia de implementação (C++/CLI)

GetModuleHandle(“Clrjit.dll”)
GetProcAddress(“getJit”)
pVTable = getJit()
pCompileMethod = VTable[0] // Armazena o método original
VTable[0] = pHookCompileMethod // Substitui o pointer para a hook function

PS: É usado o índice “0” porque a primeira entrada na VTable é o *pointer* para **compileMethod()**

Ideia de implementação

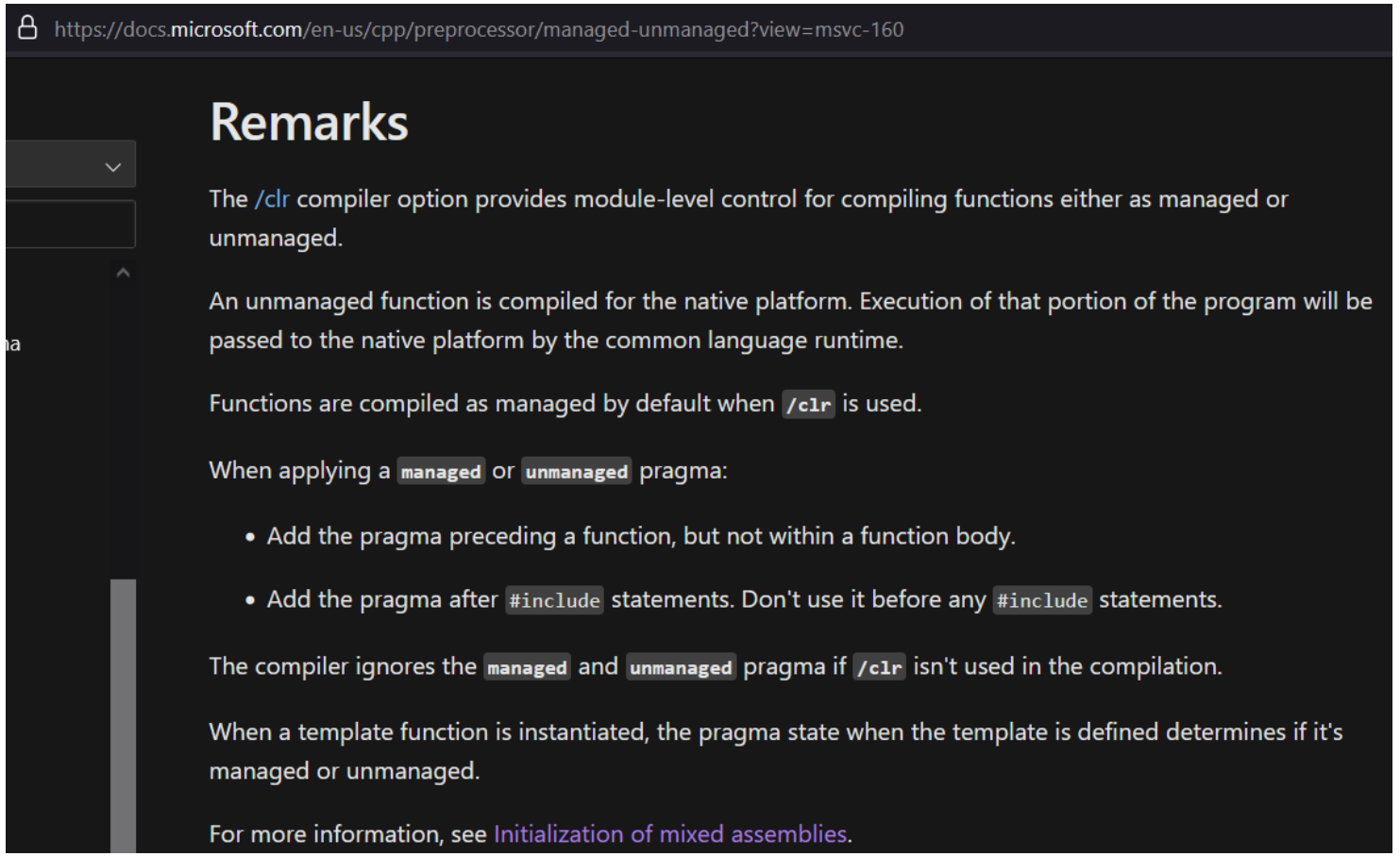



Non-hooked structure
void* clrjit.dll!getJit();
pVtable
pCompileMethod
VTable entry 1
VTable entry 2
VTable entry 3
VTable entry ...
int CILJit::compileMethod(..);
Hooked structure
void* clrjit.dll!getJit();
pVtable
pCompileMethod
VTable entry 1
VTable entry 2
VTable entry 3
VTable entry ...
int CILJit::compileMethod(..);
int hookedCompileMethod(..);

Demo

RTFM!

- https://docs.microsoft.com/en-us/cpp/preprocessor/managed-unmanaged?view=msvc-160



RTFM!

- https://docs.microsoft.com/en-us/cpp/preprocessor/managed-unmanaged?view=msvc-160

Example

C++

```
// pragma_directives_managed_unmanaged.cpp
// compile with: /clr
#include <stdio.h>

// func1 is managed
void func1() {
   System::Console::WriteLine("In managed function.");
}

// #pragma unmanaged
// push managed state on to stack and set unmanaged state
#pragma managed(push, off)

// func2 is unmanaged
void func2() {
   printf("In unmanaged function.\n");
}

// #pragma managed
#pragma managed(pop)

// main is managed
int main() {
   func1();
   func2();
}
```

Demo

Modificar o *target method*

- Para modificar o código de um método em particular, é necessário reconhecê-lo antes de ser *JITted* (ou seja, antes de entrar no processo de compilação)

- De acordo com a especificação *Common Intermediate Language* (CIL), os items dos metadados são referenciados através de um **token** de 4 bytes.
  - Exemplo de um token que identifica um método: **0x06000002**

- ICorJitInfo::getMethodDefFromMethod()
  - Obter o *pointer* do método através de VTable à semelhança do anterior

Reconhecer o *target method* (*Metadata Token*)

- Formas de obter o offset do método **getMethodDefFromMethod()** em **ICorStaticInfo**:
  1. Através do ficheiro gerado **ThunkInput.txt**
  2. Analisar **PDB** (Clrjit.pdb), pode ser obtido através do repo Microsoft Public Symbol Server

Obter offset via *ThunkInput.txt*

- https://github.com/dotnet/runtime/blob/main/src/coreclr/tools/Common/JitInterface/ThunkGenerator/ThunkInput.txt

https://github.com/dotnet/runtime/blob/main/src/coreclr/tools/Common/JitInterface/ThunkGenerator/ThunkInput.txt

```
155  FUNCTIONS
156      bool isJitIntrinsic(        CORINFO_METHOD_HANDLE       ftn        );
157      uint32_t getMethodAttribs(        CORINFO_METHOD_HANDLE       ftn        );
158      void setMethodAttribs(        CORINFO_METHOD_HANDLE       ftn,        CorInfoMethodRuntimeFlags   attribs        );
159      void getMethodSig(        CORINFO_METHOD_HANDLE      ftn,        CORINFO_SIG_INFO          *sig,        CORINFO_CLASS_HANDLE
160      bool getMethodInfo(        CORINFO_METHOD_HANDLE   ftn,        CORINFO_METHOD_INFO*    info        );
161      CorInfoInline canInline(        CORINFO_METHOD_HANDLE       callerHnd,        CORINFO_METHOD_HANDLE       calleeHnd,
162      void reportInliningDecision(CORINFO_METHOD_HANDLE inlinerHnd,        CORINFO_METHOD_HANDLE inlineeHnd,        CorInfoInline
```

(...)

```
267      void getEEInfo(CORINFO_EE_INFO* pEEInfoOut);
268      const char16_t * getJitTimeLogFilename();
269      mdMethodDef getMethodDefFromMethod(CORINFO_METHOD_HANDLE hMethod);
270      const char* getMethodName(CORINFO_METHOD_HANDLE ftn, const char **moduleName);
271      const char* getMethodNameFromMetadata(CORINFO_METHOD_HANDLE ftn, const char **className, const char **namespaceName, const ch
```

Offset **getMethodDefFromMethod()** = 269 – 156 = 113

Demo

Use cases

- Debugging
- Obfuscation
  - E.g. ConfuserEx, ByteGuard
  - Outros projetos: Jitex, SJITHook, JITM, etc.
- Malware
- Anti Reverse Engineering/Difficult Malware Analysis
- 4 Fun

Challenge – Internal CTFd

- FlagGenerator (Reverse, 200pts)
- → FlagGeneratorV2 (Reverse, 420pts)

- Sugestão de Tools:
  - dnSpy, ILSpy, IDA Pro, Ghidra, ILDASM
  - Detect-It-Easy, CFF Explorer, etc...
  - https://sharplab.io/

Referências

- http://antonioparata.blogspot.com/2018/02/analyzing-nasty-net-protection-of.html
- https://arxiv.org/ftp/arxiv/papers/1709/1709.07508.pdf
- https://black-frost.github.io/posts/allesctf_2021/
  - (Propz 0x4d5a from ALLES!)
- https://blog.matthewskelton.net/2012/01/29/advanced-call-processing-in-the-clr/
- https://blog.xpnsec.com/rundll32-your-dotnet/
- https://docs.microsoft.com/en-us/archive/blogs/abhinaba/net-just-in-time-compilation-and-warming-up-your-system
- https://docs.microsoft.com/en-us/archive/blogs/vancem/digging-into-interface-calls-in-the-net-framework-stub-based-dispatch
- https://docs.microsoft.com/en-us/cpp/dotnet/initialization-of-mixed-assemblies?view=msvc-160
- https://docs.microsoft.com/en-us/cpp/preprocessor/managed-unmanaged?view=msvc-160
- https://dotnetpad.wordpress.com/2014/06/02/what-is-ilcode-clr-cts-cls-jit/
- https://easyhook.github.io/
- https://en.wikipedia.org/wiki/List_of_CIL_instructions
- https://georgeplotnikov.github.io/articles/just-in-time-hooking.html
- https://gist.github.com/xpn/747ba8e35deee0d12ad4749b17407c26
- https://github.com/CarolEidt/coreclr/blob/master/Documentation/botr/ryujit-tutorial.md#ryujit-high-level-overview
- https://github.com/georgeplotnikov/clranalyzer
- https://github.com/maddnias/SJITHook/
- https://mattwarren.org/2017/12/15/How-does-.NET-JIT-a-method-and-Tiered-Compilation/
- https://mattwarren.org/2018/07/05/.NET-JIT-and-CLR-Joined-at-the-Hip/

Referências

- https://ntcore.com/files/disasmsil.htm
- https://pabloariasal.github.io/2017/06/10/understanding-virtual-tables/
- https://pastebin.com/dhaSjrsi
- https://pt.wikipedia.org/wiki/Common_Language_Runtime
- https://rhotav.github.io/jithooking-tr
- https://rioasmara.com/2020/08/23/ida-pro-c-vtable/
- https://rvasm.com/injetando-il-asm/
- https://software.intel.com/content/www/us/en/develop/articles/improve-the-security-of-android-applications-using-hooking-techniques-part-1.html
- https://speakerdeck.com/dotnetru/gieorghii-plotnikov-just-in-time-hooking
- https://thewover.github.io/Mixed-Assemblies/
- https://ubbecode.wordpress.com/tag/jit-compiler/
- https://www.apriorit.com/dev-blog/222-intercepting-com-calls
- https://www.c-sharpcorner.com/UploadFile/tanmayit08/unmanaged-cpp-dll-call-from-managed-C-Sharp-application/
- https://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time
- https://www.codeproject.com/Articles/781096/What-is-IL-Code-CLR-CTS-CLS-JIT
- https://www.ecma-international.org/publications-and-standards/standards/ecma-335/
- https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/
- https://www.georgeplotnikov.com/just-in-time-hooking/
- https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-1.html
- https://www.mdsec.co.uk/2020/06/detecting-and-advancing-in-memory-net-tradecraft/
- https://www.red-gate.com/simple-talk/blogs/anatomy-of-a-net-assembly-methods/
- https://xoofx.com/blog/2018/04/12/writing-managed-jit-in-csharp-with-coreclr/

About Me

- João Varelas (vrls)
  - https://vrls.ws
  - varelas@pm.me

ExtremeSTF https://xstf.pt